

```

1  #-*- coding: utf-8 -*-
2  """
3  Created on Thu Apr 18 17:38:08 2019
4
5  @author: Nate
6
7  The Generalized Metropolis algorithm removed the step-size error by an additional
8  acceptance/rejection step,
9  which adds substantial overhead. To improve on the firstorder Langevin algorithm, can
10 you devise a second-order
11 Langevin algorithm to reduce the step-size error dependence to  $(\Delta t)^2$ ?
12
13 Repeat problem 2 of HW10 using this second order Langevin algorithm.
14 """
15
16 import numpy as np
17 import matplotlib.pyplot as plt
18 import pdb
19
20 #####
21 #defining constants
22
23 N = 10000
24 alpha = 1.6875
25 g1,g2,g3,g4,g5,g6 =
26 np.random.randn(N),np.random.randn(N),np.random.randn(N),np.random.randn(N),np.random.randn(N),np.random.randn(N)
27 x_int = np.array([4,2,3,-1,-4,-4])
28 del_t = np.append(np.arange(0.001,0.01,0.003), np.arange(0.01, 0.07, 0.01))
29 r_tot = np.zeros((len(del_t), N, len(x_int)))
30 en_dat = np.zeros((len(del_t), N))
31
32 #####
33 #Function Definitions
34
35 def vel_func(r_tot):
36     r1 = np.sqrt(np.sum(r_tot[:3]**2))
37     r2 = np.sqrt(np.sum(r_tot[3:]**2))
38     v1 = -alpha*r_tot[:3]/r1
39     v2 = -alpha*r_tot[3:]/r2
40     return np.append(v1,v2)
41
42 def lan(x_int, vel_func, gau, t, N):
43     r_tot = np.zeros((N,len(x_int)))
44     vel_func1 = vel_func(x_int)
45     y0 = x_int + vel_func(x_int+t/4*vel_func1)*t/2 + gau[0]*np.sqrt(t)
46     vel_func2 = vel_func(y0)
47     r_tot[0] = y0 + t/2*vel_func(y0 + t/4*vel_func2)
48
49     for i in range(1,N):
50         vel_func1 = vel_func(r_tot[i-1])
51         Yi = r_tot[i-1] + vel_func(r_tot[i-1] + t/4*vel_func1)*t/2 + gau[i]*np.sqrt(t)
52         vel_func2 = vel_func(Yi)
53         r_tot[i] = Yi + t/2*vel_func(Yi + t/4*vel_func2)
54     return r_tot
55
56 def en(x1, y1, z1, x2, y2, z2, alpha):
57
58     r1 = np.sqrt(x1**2 + y1**2 + z1**2)
59     r2 = np.sqrt(x2**2 + y2**2 + z2**2)
60     r_diff = np.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
61
62     return alpha * (-alpha + 1/r1 + 1/r2) - 2/r1 - 2/r2 + 1/r_diff
63
64 #####
65 #####

```

```

66 #Main Loop
67
68 for i in range(len(del_t)):
69     ...
70     r_tot[i] = lan(x_int, vel_func, gau, del_t[i], N)
71     ...
72     x1 = np.array([r_tot[i,j,0] for j in range(N)])
73     y1 = np.array([r_tot[i,j,1] for j in range(N)])
74     z1 = np.array([r_tot[i,j,2] for j in range(N)])
75     x2 = np.array([r_tot[i,j,3] for j in range(N)])
76     y2 = np.array([r_tot[i,j,4] for j in range(N)])
77     z2 = np.array([r_tot[i,j,5] for j in range(N)])
78     ...
79     en_dat[i] = en(x1, y1, z1, x2, y2, z2)
80     ...
81 en_arr = np.average(en_dat, axis=1)
82 err2 = np.std(en_dat, axis=1)/np.sqrt(N/48**2)
83
84 #####
85 #####
86 #Plotting
87
88 fig1, axes1 = plt.subplots()
89 axes1.plot(del_t, en_arr, 'o', label = '2nd Order Langevin')
90 axes1.hlines(-729/256, 0, np.max(del_t), linestyle='dashed', label = 'Theoretical')
91 axes1.set_ylabel('Energy')
92 axes1.set_xlabel('Time Step Sizes $\Delta t$')
93 axes1.set_title("Energy vs $\Delta t$", va='bottom')
94 axes1.legend()
95 plt.show()
96
97
98 #####
99 #####
100
101 For problem 3, I used the fortran code provided, adding only a 4th order Forest Ruth:
102
103 !-----subprograms-----
104 ..... Subroutine schem4A(m,N,ExpTA,ExpVA,psi,ExpThalfA,ExpVhalfA,ExpTfullA,ExpVfullA)
105 !
106 !...To calculate the 4th-order decomposition scheme.
107 !
108 ..... parameter (Ndim=16384/2)
109 ..... complex*16
110         ExpTA(1),ExpVA(1),psi(1),ExpThalfA(1),ExpVhalfA(1),ExpTfullA(1),ExpVfullA(1),phi(N
111         dim)
112 !
113 ..... DO i = 1, N
114 .....     phi(i) = ExpVhalfA(i) * psi(i)
115 ..... END DO
116 ..... call fft (phi, m, 0)
117 ..... DO i = 1, N
118 .....     phi(i) = ExpThalfA(i) * phi(i)
119 ..... END DO
120 ..... call fft (phi, m, 1)
121 ..... DO i = 1, N
122 .....     psi(i) = ExpVhalfA(i) * phi(i)
123 ..... END DO
124 ..... DO i = 1, N
125 .....     phi(i) = ExpVfullA(i) * psi(i)
126 ..... END DO
127 ..... call fft (phi, m, 0)
128 ..... DO i = 1, N
129 .....     phi(i) = ExpTfullA(i) * phi(i)
130 ..... END DO
131 ..... call fft (phi, m, 1)
132 ..... DO i = 1, N
133 .....     psi(i) = ExpVfullA(i) * phi(i)
134 ..... END DO

```

```

133 .....DO i = 1, N
134 .....phi(i) = ExpVhalfA(i) * psi(i)
135 .....END DO
136 .....call fft (phi, m, 0)
137 .....DO i = 1, N
138 .....phi(i) = ExpThalfA(i) * phi(i)
139 .....END DO
140 .....call fft (phi, m, 1)
141 .....DO i = 1, N
142 .....psi(i) = ExpVhalfA(i) * phi(i)
143 .....END DO
144 !*
145 .....return
146 .....end

```